## Module Requirements: Analogue to Digital Converter

The A/D Converter system is a core part of the system, as it is used to obtain readings from the scales hardware.

### Functional Requirements

The functional requirements of the A/D module are particularly concerned with taking samples at correct intervals. This is important to guarantee constancy and reliability of the output.

#### Inputs

The A/D system requires a 5 V supply voltage to operate. The input signal must be an analogue voltage on one pin between 0 and 5 V.

#### Process

The module must operate fairly rapidly, as its operation will be performed quite frequently. A low computational complexity is highly desired.

#### Outputs

The output of the A/D converter must be a 10 bit digital value, stored in a 16 bit integer. This value must be a consistent representation of the input voltage. An input voltage of 0 V should output 0, and an input voltage of 5 volts should output 1024, thus giving a full range of possible values.

#### Timing

The sampling rate must be strictly timed such that it is regular and predictable. A suitable frequency is to be chosen, and the A/D converted must take samples at a rate as close to that frequency as is feasibly possible.

### Non-Functional Requirements

The technical nature of the A/D module means that there are the non-functional requirements are strictly secondary to functional requirements. However, during the design process several possible non-functional requirements were specified.

#### Performance

Whilst the performance of the real-time part of the module is a functional requirement, any other sections can, theoretically, take some time to execute. However, it would be preferred if computational complexity for these parts is at a feasible minimum.

#### Interfaces

Ideally the A/D system should be entirely modular. Unfortunately, the complicated nature of the system makes this unlikely; see below in Design Constraints.

#### Design Constraints

The design of the microcontroller is such that it makes it quite difficult to build the system in a completely modular fashion. In particular, the limit to only two distinct interrupts means some parts of the system that would be preferably separated cannot be. In addition, the complex real-time nature of the system makes it somewhat difficult to completely separate the A/D functionality, which is a core part of the program, from other sections.

## Conceptual Design: Analogue to Digital Converter

Designing the basic operation of the A/D module was trivial: the A/D module is initialized, the program starts the conversion, and then the program reads the result. However, the actual operation of the module was found to be more complex.

The functional requirements for the module specify real-time operation with regular sampling at a known frequency. The original design used a timer and a compare function to trigger an interrupt, which was then used to perform the A/D conversion.

However, it was quickly realized that this did not fulfill all the functional requirements. Performing the A/D conversion inside an interrupt was slow and clumsy, and there was no way of guaranteeing that the actual conversion would perform and sample at the correct rate. Thus, another solution was researched.

It was found that the CCP2 module in the PIC microcontroller can be used with a Special Event Trigger to automatically begin the A/D conversion on a timer compare. In conjunction with a Conversion Complete interrupt, this enabled the program to perform the time consuming conversion outside the interrupt, and sample at precisely the desired rate. This solved most of the previous problems.

In addition to this, in order to improve the speed of operation of the interrupt, it was decided to modify the system so the interrupt did not attempt to convert the value in the A/D register into an integer; it merely stored it for later conversion. By setting a flag indicating a new A/D value, it was possible to move this conversion into the mainline of the program, thus saving on computational complexity in the interrupt.

Shown below is a representation of how the two result registers, the Analogue to Digital Result High (ADRESH) and the Analogue to Digital Result Low (ADRESL), correspond bitwise with the integer result.

ADRESH

H1

0

0

0

0

0

0

L2

L3

L6

L5

L4

L7

L0

H0

ADRESL

L1

Integer result

L2

L3

L6

L5

L4

L7

L0

L1

H1

0

0

0

0

0

0

H0

### Assumptions Made

It was always assumed that the input was correct; that the strain gauges on the scales were working correctly. See Constraints, below.

### Constraints on A/D Converter Performance

The accuracy of the A/D converter module is highly dependent on the accuracy of the scales. If the strain gauges were incorrect, or there was some sort of hardware problem, then the A/D converter would produce incorrect results.

## Software Design: Analogue to Digital Converter

### Interface

After power up, the A/D converter hardware must be initialized. This can be performed using the initialise\_AD\_converter method in the Setup module. Once this is performed, and the global interrupts enabled, the A/D converter should automatically start sampling at a frequency of 10 Hz. To get the A/D result, the newADValue flag should be checked. If it is set, then it should be cleared, and the A/D voltage result can be obtained by calling the getVoltage function.

### Components

There are three important functions in the A/D converter module: the initialization, the interrupt, and the getVoltage method.

### initialise\_AD\_converter

This method primarily sets all the registers required to make the A/D convert module work. It starts by enabling the A/D module, and setting it to Fosc/8 timing. Then, pin A1 is chosen as the input pin. This was used rather than pin A0 because on the PIC evaluation board a potentiometer is connected to pin A0. This pin is also configured to input. The A/D converter is set to 1 analog channel with VDD and VSS references. The output registers are set to be right justified, such that it is easier to extract and use the whole 10 bit result value.

Timer 3 is enabled and set to use a x8 prescaler. This is important for the operation of the CCP2 Special Event Trigger which starts the A/D conversion. The CCP2 module itself is initialized, and set to the Special Event Trigger mode. The CCP2 compare register is set to a precalculated value to produce trigger ever 100 ms. The A/D Conversion Complete is enabled, and set to low priority.

Timer 0 is briefly enabled, and the program waits for it to time out before continuing. This is to allow the A/D converter time to start up before any conversions are attempted. Timer 0 is then disabled, and reset.

After execution of this method, the A/D converter should be fully functional. When the global interrupt flag is enabled, the A/D converter will automatically begin sampling every 100 ms, using the low priority interrupt.

### low\_ISR

The interrupt service routine for the low interrupt is used by several modules. This section only covers its use by the A/D conversion module.

The interrupt checks whether the A/D Conversion Complete interrupt is set, and if it has been triggered. If so, it performs the actions stated below, and then resets the flag.

The A/D result registers, ADRESH and ADRESL, are saved to global variables. This fundamentally takes a snapshot of the current value of the A/D result. The newADValue flag is then set. This allows the mainline of the program to detect the new value, and extract it from the stored values using getVoltage.

### getVoltage

This function is designed to return a 16 bit integer containing the 10 bit voltage value extracted from the A/D converter, as specified in the functional requirements.

At a basic level, it was designed to take the high character, place it into an integer, and left shift it 8 places such that it occupied the high byte. The low character was then to be added. Unfortunately, the technical limitations of the PIC microcontroller and C compiler made this more difficult.

The first problem was that the bit shifting in the C compiler did not comply with normal ANSI C standards: instead of performing bit shifting, it seemed to perform a rotation. Rather than try to correct this problem, it was deemed simpler to replace the bit shifting by a multiplication: in this case, a multiplication by 256.

The second problem was that converting from a character to an integer caused the high bits in the integer to be undefined. This was particularly problematic when attempting to add the low character in to the integer result. A workaround was found, involving copying the character into a temporary integer first, using a bitwise AND operation to clear the top 8 bits, and only then adding it to the result.

Thus, using the above design, the function adds the high character to the integer result, and multiplies it by 256. The low character is then converted to an integer, the high byte masked out, and then it is added to the result. Finally, the result is returned.